Đã bắt đầu vào lúc	Thứ hai, 31 Tháng mười 2022, 9:34 AM
Tình trạng	Đã hoàn thành
Hoàn thành vào lúc	Thứ hai, 31 Tháng mười 2022, 9:47 AM
Thời gian thực hiện	13 phút 16 giây
Điểm	8,00/8,00
Điểm	**10,00** của 10,00 (**100**%)

Câu hỏi 1

Chính xác

Điểm 1,00 của 1,00

In this question, you have to perform add **and delete on binary search tree**. Note that:

- When deleting a node which still have 2 children, **take the inorder successor** (smallest node of the right sub tree of that node) to replace it.

- When adding a node which has the same value as parent node, add it in the **left sub tree**.

Your task is to implement two functions: add and deleteNode. You could define one or more functions to achieve this task.

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;
#define SEPARATOR "#<ab@17943918#@>#"
template<class T>
class BinarySearchTree
{
public:
    class Node;
private:
    Node* root;
public:
    BinarySearchTree() : root(nullptr) {}
    ~BinarySearchTree()
    {
        // You have to delete all Nodes in BinaryTree. However in this task, you can ignore it.
    }

    //Helping function

    void add(T value){
        //TODO
    }

    void deleteNode(T value){
        //TODO
    }
    string inOrderRec(Node* root) {
        stringstream ss;
        if (root != nullptr) {
            ss << inOrderRec(root->pLeft);
            ss << root->value << " ";
            ss << inOrderRec(root->pRight);
        }
        return ss.str();
    }

    string inOrder(){
        return inOrderRec(this->root);
    }

    class Node
    {
    private:
        T value;
        Node* pLeft, * pRight;
        friend class BinarySearchTree<T>;
    public:
        Node(T value) : value(value), pLeft(NULL), pRight(NULL) {}
        ~Node() {}
    };
};
```

For example:

Test	Result

`BinarySearchTree<int> bst;`
`bst.add(9);`
`bst.add(2);`
`bst.add(10);`
`bst.deleteNode(9);`
`cout << bst.inOrder();`
	`2 10`

`BinarySearchTree<int> bst;`
`bst.add(9);`
`bst.add(2);`
`bst.add(10);`
`bst.add(8);`
`cout << bst.inOrder()<<endl;`
`bst.add(11);`
`bst.deleteNode(9);`
`cout << bst.inOrder();`
	`2 8 9 10`
	`2 8 10 11`

Answer: (penalty regime: 5, 10, 15, ... %)

Reset answer

```
//Helping functions
void addRec(T value,Node* preRoot, Node* root){
    if(root == nullptr){
        Node* newNode = new Node(value);
        if(value < preRoot->value) preRoot->pLeft = newNode;
        else preRoot->pRight = newNode;
        return;
    }
    if(value < root->value) addRec(value,root,root->pLeft);
    else addRec(value,root,root->pRight);
}
void add(T value){
    if(root == nullptr){
        Node* newNode = new Node(value);
        root = newNode;
        return;
    }
    if(value < root->value) addRec(value,root,root->pLeft);
    else addRec(value,root,root->pRight);
}

void deleteNodeRec(Node* root, T value,Node* preRoot){
    if (root == nullptr) return;
    if (value < root->value) deleteNodeRec(root->pLeft,value,root);
    else if(value > root->value) deleteNodeRec(root->pRight,value,root);
    else if(root->pLeft == nullptr){
        Node* tmp = root;
        if(root->value < preRoot->value) preRoot->pLeft = root->pRight;
        else preRoot->pRight = root->pRight;
        delete tmp;
        return;
    } else if(root->pRight == nullptr){
        Node* tmp = root;
        if(root->value < preRoot->value) preRoot->pLeft = root->pLeft;
        else preRoot->pRight = root->pLeft;
        delete tmp;
        return;
    } else{
        Node* tmp = root->pLeft;
        while(tmp->pRight != nullptr) tmp = tmp->pRight;
        int newValue = tmp->value;
        deleteNodeRec(root->pLeft,newValue,root);
        root->value = newValue;
    }
```

```
45  }
46  void deleteNode(T value){
47      if(value < root->value) deleteNodeRec(root->pLeft,value,root);
48      else if(value > root->value) deleteNodeRec(root->pRight,value,root);
49      else if(root->pLeft == nullptr){
50          Node* tmp = root;
51          root = root->pRight;
52          delete tmp;
53          return;
54      } else if(root->pRight == nullptr){
55          Node* tmp = root;
56          root = root->pLeft;
57          delete tmp;
58          return;
59      } else{
60          Node* tmp = root->pLeft;
61          while(tmp->pRight != nullptr) tmp = tmp->pRight;
62          int newValue = tmp->value;
```

	Test	Expected	Got	

✔	BinarySearchTree<int> bst;
	bst.add(9);
	bst.add(2);
	bst.add(10);
	bst.deleteNode(9);
	cout << bst.inOrder();
		2 10	2 10	✔

✔	BinarySearchTree<int> bst;
	bst.add(9);
	bst.add(2);
	bst.add(10);
	bst.add(8);
	cout << bst.inOrder()<<endl;
	bst.add(11);
	bst.deleteNode(9);
	cout << bst.inOrder();
		2 8 9 10
		2 8 10 11
			2 8 9 10
			2 8 10 11
				✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 2

Chính xác

Điểm 1,00 của 1,00

Given class **BinarySearchTree**, you need to finish method getMin() and getMax() in this question.

```
#include <iostream>
#include <string>
#include <sstream>

using namespace std;

template<class T>
class BinarySearchTree
{
public:
    class Node;

private:
    Node* root;

public:
    BinarySearchTree() : root(nullptr) {}
    ~BinarySearchTree()
    {
        // You have to delete all Nodes in BinaryTree. However in this task, you can ignore it.
    }

    class Node
    {
    private:
        T value;
        Node* pLeft, * pRight;
        friend class BinarySearchTree<T>;

    public:
        Node(T value) : value(value), pLeft(NULL), pRight(NULL) {}
        ~Node() {}
    };
    Node* addRec(Node* root, T value);
    void add(T value) ;
    // STUDENT ANSWER BEGIN

    // STUDENT ANSWER END
};
```


For example:

Test	Result

`BinarySearchTree<int> bst;`
`for (int i = 0; i < 10; ++i) {`
`  bst.add(i);`
`}`
`cout << bst.getMin() << endl;`
`cout << bst.getMax() << endl;`
	`0`
	`9`

Answer: (penalty regime: 5, 10, 15, ... %)

Reset answer

```
// STUDENT ANSWER BEGIN
// You can define other functions here to help you.

T getMin() {
    //TODO: return the minimum values of nodes in the tree.
    Node* tmp = root;
    while(tmp->pLeft != nullptr) tmp = tmp->pLeft;
    return tmp->value;
}

T getMax() {
    //TODO: return the maximum values of nodes in the tree.
    Node* tmp = root;
    while(tmp->pRight != nullptr) tmp = tmp->pRight;
    return tmp->value;

}

// STUDENT ANSWER END
```

	Test	Expected	Got	

✔	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 10; ++i) {`
	`  bst.add(i);`
	`}`
	`cout << bst.getMin() << endl;`
	`cout << bst.getMax() << endl;`
		0
		9
			0
			9
				✔

✔	`int values[] = { 66,60,84,67,21,45,62,1,80,35 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 10; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.getMin() << endl;`
	`cout << bst.getMax() << endl;`
		1
		84
			1
			84
				✔

✔	`int values[] = { 38,0,98,38,99,67,19,70,55,6 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 10; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.getMin() << endl;`
	`cout << bst.getMax() << endl;`
		0
		99
			0
			99
				✔

✔	`int values[] = { 34,81,73,48,66,91,19,84,78,79 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 10; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.getMin() << endl;`
	`cout << bst.getMax() << endl;`
		19
		91
			19
			91
				✔

	Test	Expected	Got	

✔	`int values[] = { 94,61,75,36,34,58,62,74,54,90 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 10; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.getMin() << endl;`
	`cout << bst.getMax() << endl;`
		34
		94
			34
			94
				✔

✔	`int values[] = { 32,0,2,84,34,78,70,60,95,71,26,62,0,22,95 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 15; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.getMin() << endl;`
	`cout << bst.getMax() << endl;`
		0
		95
			0
			95
				✔

✔	`int values[] = { 53,24,32,40,80,47,81,88,42,29,31,91,77,73,90 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 15; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.getMin() << endl;`
	`cout << bst.getMax() << endl;`
		24
		91
			24
			91
				✔

✔	`int values[] = { 32,19,23,33,76,1,37,53,18,89,28,1,77,52,17 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 15; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.getMin() << endl;`
	`cout << bst.getMax() << endl;`
		1
		89
			1
			89
				✔

✔	`int values[] = { 25,29,57,30,62,56,60,55,88,56,70,83,56,75,17 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 15; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.getMin() << endl;`
	`cout << bst.getMax() << endl;`
		17
		88
			17
			88
				✔

✔	`int values[] = { 75,13,83,83,30,40,10,86,17,21,45,22,22,72,63 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 15; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.getMin() << endl;`
	`cout << bst.getMax() << endl;`
		10
		86
			10
			86
				✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 3

Chính xác

Điểm 1,00 của 1,00

Given class **BinarySearchTree**, you need to finish method **find(i)** to check whether value i is in the tree or not; method **sum(l,r)** to calculate sum of all all elements v in the tree that has value greater than or equal to l and less than or equal to r.

```
#include <iostream>
#include <string>
#include <sstream>

using namespace std;

template<class T>
class BinarySearchTree
{
public:
    class Node;

private:
    Node* root;

public:
    BinarySearchTree() : root(nullptr) {}
    ~BinarySearchTree()
    {
        // You have to delete all Nodes in BinaryTree. However in this task, you can ignore it.
    }

    class Node
    {
    private:
        T value;
        Node* pLeft, * pRight;
        friend class BinarySearchTree<T>;

    public:
        Node(T value) : value(value), pLeft(NULL), pRight(NULL) {}
        ~Node() {}
    };
    Node* addRec(Node* root, T value);
    void add(T value) ;
    // STUDENT ANSWER BEGIN

    // STUDENT ANSWER END
};
```


For example:

Test	Result

BinarySearchTree<int> bst;
for (int i = 0; i < 10; ++i) {
bst.add(i);
}
cout << bst.find(7) << endl;
cout << bst.sum(0, 4) << endl
	1
	10

Answer: (penalty regime: 5, 10, 15, ... %)

Reset answer

```
1 // STUDENT ANSWER BEGIN
2 // You can define other functions here to help you.
```

```
bool find(T i, Node* root){
    if(root == nullptr) return false;
    if(i < root->value) return find(i,root->pLeft);
    if(i > root->value) return find(i,root->pRight);
    return true;
}
bool find(T i) {
    // TODO: return true if value i is in the tree; otherwise, return false.
    return find(i,root);
}
Node* findNode(T i, Node* root){
    if(i < root->value) return findNode(i,root->pLeft);
    if(i > root->value) return findNode(i,root->pRight);
    return root;
}
T sum(T l, T r) {
    // TODO: return the sum of all element in the tree has value in range [l,r].
    Node* tmp = root;
    int sum = 0;
    //bool contain = false;
```

	Test	Expected	Got	

✔	BinarySearchTree<int> bst;
	for (int i = 0; i < 10; ++i) {
	bst.add(i);
	}
	cout << bst.find(7) << endl;
	cout << bst.sum(0, 4) << endl
		1
		10
			1
			10
				✔

✔	int values[] = { 66,60,84,67,21,45,62,1,80,35 };
	BinarySearchTree<int> bst;
	for (int i = 0; i < 10; ++i) {
	bst.add(values[i]);
	}
	
	cout << bst.find(5) << endl;
	cout << bst.sum(10, 40);
		0
		56
			0
			56
				✔

✔	int values[] = { 38,0,98,38,99,67,19,70,55,6 };
	BinarySearchTree<int> bst;
	for (int i = 0; i < 10; ++i) {
	bst.add(values[i]);
	}
	
	cout << bst.find(5) << endl;
	cout << bst.sum(10, 40);
		0
		95
			0
			95
				✔

✔	int values[] = { 34,81,73,48,66,91,19,84,78,79 };
	BinarySearchTree<int> bst;
	for (int i = 0; i < 10; ++i) {
	bst.add(values[i]);
	}
	
	cout << bst.find(5) << endl;
	cout << bst.sum(10, 40);
		0
		53
			0
			53
				✔

✔	int values[] = { 94,61,75,36,34,58,62,74,54,90 };
	BinarySearchTree<int> bst;
	for (int i = 0; i < 10; ++i) {
	bst.add(values[i]);
	}
	
	cout << bst.find(34) << endl;
	cout << bst.sum(10, 40);
		1
		70
			1
			70
				✔

	Test	Expected	Got	

✔	`int values[] = { 32,0,2,84,34,78,70,60,95,71,26,62,0,22,95 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 15; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.find(34) << endl;`
	`cout << bst.sum(10, 40);`
		1
		114
			1
			114
				✔

✔	`int values[] = { 53,24,32,40,80,47,81,88,42,29,31,91,77,73,90 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 15; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.find(34) << endl;`
	`cout << bst.sum(10, 40);`
		0
		156
			0
			156
				✔

✔	`int values[] = { 32,19,23,33,76,1,37,53,18,89,28,1,77,52,17 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 15; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.find(34) << endl;`
	`cout << bst.sum(10, 40);`
		0
		207
			0
			207
				✔

✔	`int values[] = { 25,29,57,30,62,56,60,55,88,56,70,83,56,75,17 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 15; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.find(34) << endl;`
	`cout << bst.sum(10, 40);`
		0
		101
			0
			101
				✔

✔	`int values[] = { 75,13,83,83,30,40,10,86,17,21,45,22,22,72,63 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 15; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.find(34) << endl;`
	`cout << bst.sum(10, 40);`
		0
		175
			0
			175
				✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 4

Chính xác

Điểm 1,00 của 1,00

Class `BSTNode` is used to store a node in binary search tree, described on the following:

```
class BSTNode {
public:
    int val;
    BSTNode *left;
    BSTNode *right;
    BSTNode() {
        this->left = this->right = nullptr;
    }
    BSTNode(int val) {
        this->val = val;
        this->left = this->right = nullptr;
    }
    BSTNode(int val, BSTNode*& left, BSTNode*& right) {
        this->val = val;
        this->left = left;
        this->right = right;
    }
};
```

Where `val` is the value of node, `left` and `right` are the pointers to the left node and right node of it, respectively. If a repeated value is inserted to the tree, it will be inserted to the left subtree.

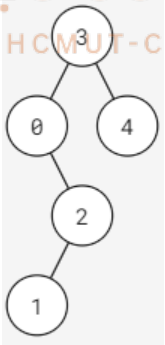
Request: Implement function:

```
vector<int> levelAlterTraverse(BSTNode* root);
```

Where `root` is the root node of given binary search tree (this tree has between 0 and 100000 elements). This function returns the values of the nodes in each level, alternating from going left-to-right and right-to-left..

Example:

Given a binary search tree in the following:



In the first level, we should traverse from left to right (order: `3`) and in the second level, we traverse from right to left (order: `4, 0`). After traversing all the nodes, the result should be `[3, 4, 0, 2, 1]`.

Note: In this exercise, the libraries `iostream`, `vector`, `stack`, `queue`, `algorithm` and `using namespace std` are used. You can write helper functions; however, you are not allowed to use other libraries.

For example:

Test	Result

`int arr[] = {0, 3, 5, 1, 2, 4};`
`BSTNode* root = BSTNode::createBSTree(arr, arr + sizeof(arr)/sizeof(int));`
`printVector(levelAlterTraverse(root));`
`BSTNode::deleteTree(root);`
	`[0, 3, 1, 5, 4, 2]`

Answer: (penalty regime: 0, 0, 0, 5, 10, ... %)

Reset answer

```
vector<int> levelAlterTraverse(BSTNode* root) {
    vector<BSTNode*> q;
    vector<int> ans;
    if(root == nullptr) return ans;
    bool L_R = false;
    q.push_back(root);
    while(!q.empty()){
        vector<BSTNode*>::iterator it = q.begin();
        while(it != q.end()){
            ans.push_back((*it)->val);
            it++;
        }
        vector<BSTNode*> aux;
        while(!q.empty()){
            if(L_R){
                if(q.back()->left) aux.push_back(q.back()->left);
                if(q.back()->right) aux.push_back(q.back()->right);
            }
            else{
                if(q.back()->right) aux.push_back(q.back()->right);
                if(q.back()->left) aux.push_back(q.back()->left);
            }
```

	Test	Expected	Got	

✔	`int arr[] = {0, 3, 5, 1, 2, 4};`
	`BSTNode* root = BSTNode::createBSTree(arr, arr + sizeof(arr)/sizeof(int));`
	`printVector(levelAlterTraverse(root));`
	`BSTNode::deleteTree(root);`
		`[0, 3, 1, 5, 4, 2]`	`[0, 3, 1, 5, 4, 2]`	✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 5

Chính xác

Điểm 1,00 của 1,00

Class **BTNode** is used to store a node in binary search tree, described on the following:

```
class BTNode {
    public:
        int val;
        BTNode *left;
        BTNode *right;
        BTNode() {
            this->left = this->right = NULL;
        }
        BTNode(int val) {
            this->val = val;
            this->left = this->right = NULL;
        }
        BTNode(int val, BTNode*& left, BTNode*& right) {
            this->val = val;
            this->left = left;
            this->right = right;
        }
};
```

Where `val` is the value of node (non-negative integer), `left` and `right` are the pointers to the left node and right node of it, respectively.

Request: Implement function:

```
int rangeCount(BTNode* root, int lo, int hi);
```

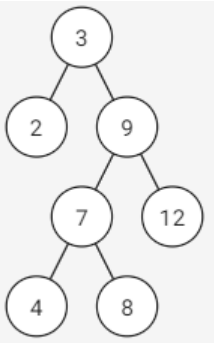
Where `root` is the root node of given binary search tree (this tree has between 0 and 100000 elements), `lo` and `hi` are 2 positives integer and `lo ≤ hi`. This function returns the number of all nodes whose values are between `[lo, hi]` in this binary search tree.

More information:

- If a node has `val` which is equal to its ancestor's, it is in the right subtree of its ancestor.

Example:

Given a binary search tree in the following:



With `lo=5, hi=10`, all the nodes satisfied are node `9, 7, 8`; there fore, the result is `3`.

Note: In this exercise, the libraries `iostream`, `stack`, `queue`, `utility` and `using namespace std` are used. You can write helper functions; however, you are not allowed to use other libraries.

For example:

Test	Result

int value[] = {3,2,9,7,12,4,8};
int lo = 5, hi = 10;
BTNode* root = BTNode::createBSTree(value, value + sizeof(value)/sizeof(int));
cout << rangeCount(root, lo, hi);
	3

int value[] = {1167,2381,577,2568,124,1519,234,1679,2696,2359};
int lo = 500, hi = 2000;
BTNode* root = BTNode::createBSTree(value, value + sizeof(value)/sizeof(int));
cout << rangeCount(root, lo, hi);
	4

Answer: (penalty regime: 0 %)

Reset answer

```
bool find(int i, BTNode * root){
    if(root == nullptr) return false;
    if(i < root->val) return find(i,root->left);
    if(i > root->val) return find(i,root->right);
    return true;
}

BTNode * findNode(int i, BTNode * root){
    if(i < root->val) return findNode(i,root->left);
    if(i > root->val) return findNode(i,root->right);
    return root;
}
int rangeCount(BTNode* root, int lo, int hi) {
    BTNode * tmp = root;
    int sum = 0;
    //bool contain = false;
    for(int i = lo; i <= hi; i++){
        while(find(i,tmp)){
            sum++;
            tmp = findNode(i,tmp);
            tmp = tmp->right;
        }
        tmp = root;
    }
    return sum;
}
```

	Test	Expected	Got	

✔	`int value[] = {3,2,9,7,12,4,8};`
	`int lo = 5, hi = 10;`
	`BTNode* root = BTNode::createBSTree(value, value + sizeof(value)/sizeof(int));`
	`cout << rangeCount(root, lo, hi);`
		3	3	✔

✔	`int value[] = {1167,2381,577,2568,124,1519,234,1679,2696,2359};`
	`int lo = 500, hi = 2000;`
	`BTNode* root = BTNode::createBSTree(value, value + sizeof(value)/sizeof(int));`
	`cout << rangeCount(root, lo, hi);`
		4	4	✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 6

Chính xác

Điểm 1,00 của 1,00

Class `BSTNode` is used to store a node in binary search tree, described on the following:

```
class BSTNode {
public:
    int val;
    BSTNode *left;
    BSTNode *right;
    BSTNode() {
        this->left = this->right = nullptr;
    }
    BSTNode(int val) {
        this->val = val;
        this->left = this->right = nullptr;
    }
    BSTNode(int val, BSTNode*& left, BSTNode*& right) {
        this->val = val;
        this->left = left;
        this->right = right;
    }
};
```

Where `val` is the value of node, `left` and `right` are the pointers to the left node and right node of it, respectively. If a repeated value is inserted to the tree, it will be inserted to the left subtree.

Request: Implement function:

```
int singleChild(BSTNode* root);
```

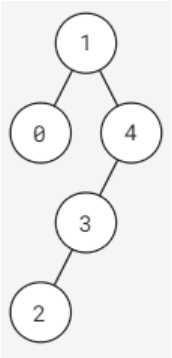
Where `root` is the root node of given binary search tree (this tree has between 0 and 100000 elements). This function returns the number of single children in the tree.

More information:

- A node is called a **single child** if its parent has only one child.

Example:

Given a binary search tree in the following:



There are `2` single children: node `2` and node `3`.

Note: In this exercise, the libraries `iostream` and `using namespace std` are used. You can write helper functions; however, you are not allowed to use other libraries.

For example:

Test	Result

`int arr[] = {0, 3, 5, 1, 2, 4};`
`BSTNode* root = BSTNode::createBSTree(arr, arr + sizeof(arr)/sizeof(int));`
`cout << singleChild(root);`
`BSTNode::deleteTree(root);`
	3

Answer: (penalty regime: 0, 0, 0, 5, 10, ... %)

Reset answer

```
int singleChild(BSTNode* root) {
    if(root == nullptr) return 0;
    if(root->left && root->right) return singleChild(root->left) + singleChild(root->right);
    else if(!root->left && !root->right) return 0;
    return 1 + singleChild(root->left) + singleChild(root->right);
}
```

	Test	Expected	Got	
✔	int arr[] = {0, 3, 5, 1, 2, 4}; BSTNode* root = BSTNode::createBSTree(arr, arr + sizeof(arr)/sizeof(int)); cout << singleChild(root); BSTNode::deleteTree(root);	3	3	✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 7

Chính xác

Điểm 1,00 của 1,00

Class `BSTNode` is used to store a node in binary search tree, described on the following:

```
class BSTNode {
public:
    int val;
    BSTNode *left;
    BSTNode *right;
    BSTNode() {
        this->left = this->right = nullptr;
    }
    BSTNode(int val) {
        this->val = val;
        this->left = this->right = nullptr;
    }
    BSTNode(int val, BSTNode*& left, BSTNode*& right) {
        this->val = val;
        this->left = left;
        this->right = right;
    }
};
```

Where `val` is the value of node, `left` and `right` are the pointers to the left node and right node of it, respectively. If a repeated value is inserted to the tree, it will be inserted to the left subtree.

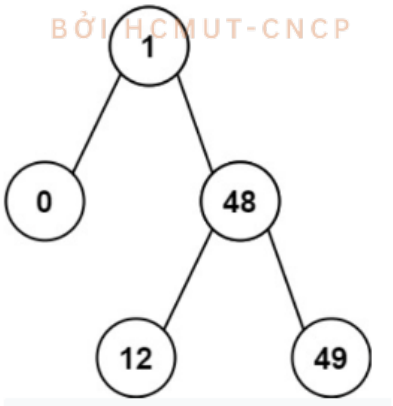
Request: Implement function:

```
int kthSmallest(BSTNode* root, int k);
```

Where `root` is the root node of given binary search tree (this tree has `n` elements) and `k` satisfy: `1 <= k <= n <= 100000`. This function returns the `k`-th smallest value in the tree.

Example:

Given a binary search tree in the following:



With `k = 2`, the result should be `1`.

Note: In this exercise, the libraries `iostream`, `vector`, `stack`, `queue`, `algorithm`, `climits` and `using namespace std` are used. You can write helper functions; however, you are not allowed to use other libraries.

For example:

Test	Result

int arr[] = {6, 9, 2, 13, 0, 20};
int k = 2;
BSTNode* root = BSTNode::createBSTree(arr, arr + sizeof(arr)/sizeof(int));
cout << kthSmallest(root, k);
BSTNode::deleteTree(root);
	2

Answer: (penalty regime: 0, 0, 0, 5, 10, ... %)

Reset answer

```
vector<int> inOrder(BSTNode* root,vector<int> ans){
    if(root == nullptr) {
        return ans;
    }
    vector<int> l,r;
    //if(root->left) l = inOrder(root->left,ans);
    //if(root->right) r = inOrder(root->right,ans);
    l = inOrder(root->left,ans);
    r = inOrder(root->right,ans);
    ans.insert(ans.end(),l.begin(),l.end());
    ans.push_back(root->val);
    ans.insert(ans.end(),r.begin(),r.end());
    return ans;
}
int kthSmallest(BSTNode* root, int k) {
    vector<int> in;
    in = inOrder(root,in);
    return in[k-1];
}
```


	Test	Expected	Got	

✔	int arr[] = {6, 9, 2, 13, 0, 20};
	int k = 2;
	BSTNode* root = BSTNode::createBSTree(arr, arr + sizeof(arr)/sizeof(int));
	cout << kthSmallest(root, k);
	BSTNode::deleteTree(root);
		2	2	✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 8

Chính xác

Điểm 1,00 của 1,00

Class `BSTNode` is used to store a node in binary search tree, described on the following:

```
class BSTNode {
public:
    int val;
    BSTNode *left;
    BSTNode *right;
    BSTNode() {
        this->left = this->right = nullptr;
    }
    BSTNode(int val) {
        this->val = val;
        this->left = this->right = nullptr;
    }
    BSTNode(int val, BSTNode*& left, BSTNode*& right) {
        this->val = val;
        this->left = left;
        this->right = right;
    }
};
```

Where `val` is the value of node, `left` and `right` are the pointers to the left node and right node of it, respectively. If a repeated value is inserted to the tree, it will be inserted to the left subtree.

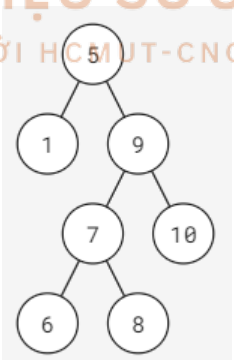
Request: Implement function:

```
BSTNode* subtreeWithRange(BSTNode* root, int lo, int hi);
```

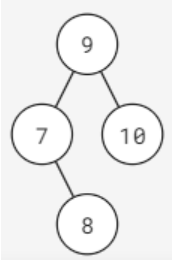
Where `root` is the root node of given binary search tree (this tree has between 0 and 100000 elements). This function returns the binary search tree after deleting all nodes whose values are outside the range `[lo, hi]` (inclusive).

Example:

Given a binary search tree in the following:



With `lo = 7` and `hi = 10`, the result should be:



Note: In this exercise, the libraries `iostream` and `using namespace std` are used. You can write helper functions; however, you are not allowed to use other libraries.

For example:

Test	Result

int arr[] = {0, 3, 5, 1, 2, 4};
int lo = 1, hi = 3;
BSTNode* root = BSTNode::createBSTree(arr, arr + sizeof(arr)/sizeof(int));
root = subtreeWithRange(root, lo, hi);
BSTNode::printPreorder(root);
BSTNode::deleteTree(root);
	3 1 2

Answer: (penalty regime: 0, 0, 0, 5, 10, ... %)

Reset answer

```
BSTNode* deleteBSTNodeRec(BSTNode* root, int val,BSTNode* preRoot,BSTNode* originRoot){
    if (root == nullptr) return originRoot;
    if (val < root->val) return deleteBSTNodeRec(root->left,val,root,originRoot);
    else if(val > root->val) return deleteBSTNodeRec(root->right,val,root,originRoot);
    else if(root == originRoot){
        if(originRoot->left == nullptr){
            BSTNode* tmp = originRoot;
            originRoot = originRoot->right;
            delete tmp;
            return originRoot;
        }
        if(originRoot->right == nullptr){
            BSTNode* tmp = originRoot;
            originRoot = originRoot->left;
            delete tmp;
            return originRoot;
        }
        BSTNode* tmp = originRoot->right;
        while(tmp->left != nullptr) tmp = tmp->left;
        int newval = tmp->val;
        deleteBSTNodeRec(originRoot->right,newval,originRoot,originRoot);
        originRoot->val = newval;
```


	Test	Expected	Got	

✔	int arr[] = {0, 3, 5, 1, 2, 4};
	int lo = 1, hi = 3;
	BSTNode* root = BSTNode::createBSTree(arr, arr + sizeof(arr)/sizeof(int));
	root = subtreeWithRange(root, lo, hi);
	BSTNode::printPreorder(root);
	BSTNode::deleteTree(root);
		3 1 2	3 1 2	✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

HCMUT

MyBK

BKSI

LIÊN HỆ

268 Lý Thường Kiệt, P.14, Q.10, TP.HCM

(028) 38 651 670 - (028) 38 647 256 (Ext: 5258, 5234)

elearning@hcmut.edu.vn